REMARKS

The changes to the application and claims are believed to be merely correction of typographical errors and clarification of the application. Therefore, Applicants believe that no new matter has been added.

7/9/01

Eamon J. Wall, Attorney
Registration No. 39,414
(732) 530-9404

Thomason, Moser & Patterson LLP
Attorneys at Law
595 Shrewsbury Avenue
Suite 100
Shrewsbury, New Jersey 07702

APPENDIX I

MARKED-UP VERSION OF AMENDED SPECIFICATION PARAGRAPHS

Mark-up of paragraph [0008] beginning on page 3:

An API according to an embodiment of the invention comprises first and second data structures associated with a network interface in communication with a network, the first and second data structures being mapped to an operating system and a network application, wherein: packets to be passed from the operating system to the network application are stored in a buffer and referenced via respective pointers within the first data structure, the first data structure pointers being inserted into the first data structure by the operating system prior to network layer processing, the first data structure pointers being removed by the network application, insertion and removal of the first data structure pointers being asynchronous with respect to each other; and packets to be processed as received packets by the network layer of the operating system are stored in a buffer and referenced via respective pointers within the second data structure, the [nsecond] second data structure pointers being inserted into the second data structure by the network application, the second data structure pointers being removed by said operating system, insertion and removal of the second data structure pointers being asynchronous with respect to each other.

Mark-up of paragraph [0009] beginning on page 4:

An API according to another embodiment of the invention for network applications, which applications can process packets whose source and destination nodes are nodes different from that where the application runs, the API comprising a primitive for creating a first and a second data structures associated with a specified network interface, if the data structures do not exist, and mapping the data structures both to the operating system and a specified network application, wherein the specified network interface receives and sends packets from and to a network, each [the] packet is stored in a buffer mapped both to the operating system and the specified network application, the operating

system inserts into and the specified network application may remove from the first data structure a pointer to each buffer containing a packet that the operating system's network layer outputs to the specified network interface, before the network interface sends the packets, the insertions and removals being asynchronous with respect to each other, and the specified network application may insert into and the operating system removes from the second data structure a pointer to each buffer containing a packet that the specified network interface sends to the network, the insertions and removals being asynchronous with respect to each other.

Mark-up of paragraph [0024] on page 8:

A circular queue comprises a first data structure and a second data structure. Specifically, first and second data structures comprise a detour and revert queue respectively which are mapped to both the operating system 102 and network application 108. More specifically, an input tap 118 comprises a detour queue 104A and a revert queue 106A. Detour queue 104A is coupled to the network interface 114 and network application 108. Revert queue 106A is coupled to network application 108 and TCP/IP [queue] implementation 112.

Mark-up of paragraph [0025] on page 8:

An output tap 120 comprises a detour queue 104B and a revert queue 106B. Detour queue 104B is coupled to TCP/IP implementation 112 and network application 108. Revert queue 106B is coupled to network application 108 and [the plurality of] to the network interface 114.

Mark-up of paragraph [0029] on page 9:

As incoming packets arrive, the packets are stored in memory. Data structures are created wherein pointers are placed in the data structures by the operating system 102. Specifically, pointers that point to the location of the incoming packets in memory are placed in detour queue 104A and taken out of

detour queue 104A by the network application 108. [After the packets have been processed by network application 108, pointers are placed in revert queue 106A by network application 108 and taken out by operating system 102 for transmission out to the network via a respective device i.e., host application 110, TCP/IP implementation 112 or directly to the plurality of network interface 114.] After the network application 108 has processed a packet, the network application 108 can insert a pointer 410 to the packet buffer 402 in one of three queues. First, the network application 108 can insert the pointer 410 in revert queue 106A. In this case, the operating system's TCP/IP implementation processes the packet as received from the network 116. Second, the network application 108 can insert the pointer 410 in revert queue 106B. In this case, the network interface 114 sends the packet to the network 116. Third, the network application 108 can insert the pointer 410 in the mbuf_dealloc queue 304. In this case, the operating system deallocates the packet buffer 402.

Mark-up of paragraph [0031] on page 10:

FIG. 2 depicts a program segment written in the C programming language including an API according to the invention. Specifically, program 200 is written in the C programming language and comprises a circular queue representation 202, inline functions for enqueueing 204 and dequeueing 206 a pointer to or from a circular queue, system calls including mbuf_map 208, mbuf_unmap 210, mbuff_pull 212, mbuf_push 214, interface_tap 216 and interface_untap 218. For a better understanding of the invention, program 200 should be read in view of FIG. 3 and FIG. 1 together.

Mark-up of paragraph [0033] on page 10:

As previously discussed above with reference to FIG. 1, all mbufs 402 are allocated from a single unpageable region. NetTap's "mbuf_map" primitive 208 maps this region to the application's address space. [Circular queue representation routine 202 allow circular queues to be created.] Specifically, the primitive "mbuf_map" 208 creates two circular queues, "mbuf_alloc" 302 and

"mbuf_dealloc" 304, and maps them to the application's address space, as shown in FIG. 3. The "mbuf_unmap" primitive 210 unmaps the mbuf region and destroys the application's mbuf_alloc 302 and mbuf_dealloc 304 queues.

Mark-up of paragraph [0034] beginning on page 10:

NetTap network applications 108 allocate mbufs by dequeueing the mbufs' pointers 410 from mbuf_alloc 302 using the dequeue routine 206, and deallocate mbufs by enqueueing the mbufs' pointers 410 in mbuf_dealloc 304 using the enqueue routine 204. The mbuf_map primitive 208 includes an argument, "mbuf_prealloc" (not shown), that specifies the minimum number of mbuf pointers 410 that should be enqueued in the application's mbuf_alloc queue 302. The operating system 102 asynchronously replenishes queue 302, making mbufs 402 available to the network application 108. When necessary (e.g., mbuf_alloc 302 is empty), however, applications 108 may use the mbuf_pull primitive 212 to force the system 102 to enqueue a specified (strictly positive) number of mbuf pointers 410 synchronously into mbuf_alloc 302, subject to a timeout interval specified in microseconds (infinite if set to 0). The system 102 asynchronously dequeues pointers 410 from the applications' 108 mbuf_dealloc queues 304 and deallocates the respective mbufs. If necessary (e.g., mbuf_dealloc 304 is full), however, an application [6] 108 may use the "mbuf_push" primitive 214, with the tap descriptor argument equal to -1, to force the system 102 to dequeue any pointers 410 from mbuf_dealloc 304 synchronously and deallocate the respective buffers. The primitives mbuf_map 208, mbuf_unmap 210, mbuf_push 212, and mbuf_pull 214 return 0 if successful, or an error code otherwise.

Mark-up of paragraph [0038] on page 12:

Conversely, if the interface_tap primitive's 216 mode is TAP_OUTPUT, a host application 110 may use the system's regular API to send packets, but the system enqueues in detour 104B packet pointers 410 that TCP/IP implementation 112 would normally pass to [one of] the network interface 114 for transmission to the network 116. The system 102 dequeues packet pointers 410

from revert 106B and passes them to [one of] the network interface 114 for transmission to the network 116.

Mark-up of paragraph [0039] beginning on page 12:

Finally, if the interface_tap primitive's 216 mode is TAP_BYPASS, host applications 110 cannot use the system's regular API to send or receive packets via [one of] the network interface 114 and the system's firewalling and IP forwarding become inoperative on the [plurality of] network interface 114. The system 102 enqueues in detour 104A pointers 410 to packets received by [one of] the network interface 114 from the network 116, and dequeues packet pointers 410 from revert 106B and passes them to the network interface for transmission to the network 116. As shown in FIG. 1, a given network interface 114 can have both TAP_INPUT 118 and TAP_OUTPUT 120 taps. However, a given network interface [from the network interface 114] cannot have a TAP_INPUT 118 or TAP_OUTPUT 120 tap and also have a TAP_BYPASS tap (not shown).

Mark-up of paragraph [0042] on page 14:

Conversely, network applications 108 output packets by enqueuing in revert queues 106A or 106B pointers 410 to the mbufs containing the packets. The system 102 asynchronously dequeues mbuf pointers 410 from revert queues 106A and 106B and processes the respective packets (in the TAP_INPUT case, passes them to the IP input [implementation 112] queue of the system's TCP/IP implementation 112; in the TAP_OUTPUT and TAP_BYPASS cases, passes them to [one of] the network interface 114 for transmission to the network 116). If necessary (e.g., a revert queue 106B is full), however, applications 108 may use the mbuf_push primitive 214 to wait for the system 102 to dequeue a specified number of pointers 410 from a specified revert queue 106A or 106B.

APPENDIX II

Marked-up Claims

4. (Amended) The API of claim 3, wherein:

[in the case of] if said first and a second data structures are not [being] associated with the network interface, the operating system automatically passes packets received from the network by the network interface to the operating system's network layer, for processing, and automatically passes packets output by the operating system's network layer to the network interface, for sending to the network. [packets to be passed between the network and the network interface are processed by the operating system network layer.]

6. (Amended) The API of claim 1, further comprising a primitive for creating said first and a second data structures mapped both to said operating system and said network application, wherein:

[non-network packets] allocated buffers to be passed from the operating system to the network application are [stored in a buffer and] referenced via respective pointers within said first data structure, said first data structure pointers being inserted into said first data structure by said operating system, said first data structure pointers being removed by said network application; and

[non-network packets] deallocated buffers to be passed from said network application to said operating system are [stored in a buffer and] referenced via respective pointers within said second data structure, said second data structure pointers being inserted into said second data structure by said network application, said second data structure pointers being removed by said operating system.

9. (Amended) The API of claim 1 wherein other network applications [are prevented from accessing] do not access a buffer from the time said network application removes a pointer to said buffer from said first data structure and inserts a pointer to said buffer into said second data structure.

10. (Amended) The API of claim 9, wherein each buffer contains an identifier of [any] a network application having exclusive use of the buffer.

16. (Amended) The API of claim 6 wherein other network applications [are prevented from accessing] do not access a buffer from the time said network application removes a pointer to said buffer from said first data structure and inserts a pointer to said buffer into said second data structure.